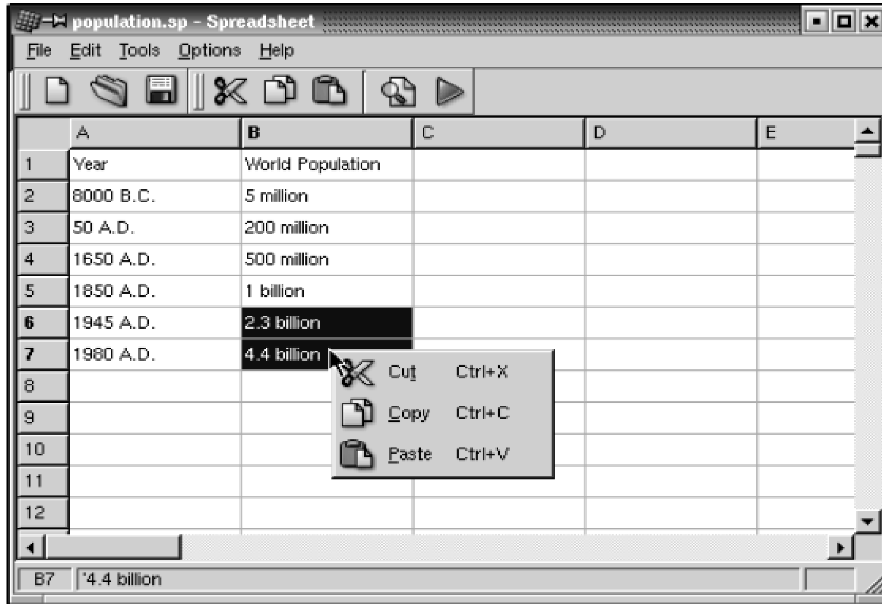
3

- *QmainWindow dan alt sınıf oluşturulması*
- *Menüler ve araç çubukları*
- *Dosya (File) Menüsü*
- *Durum çubuğunun kullanılışı*
- *Diyalogların istimali*
- *Ayarların kaydedilmesi*
- *Birden fazla dosyanın aynı anda açılması*

Ana Pencerelerin Oluşturulması

Bu bölümde Qt kullanarak nasıl *ana pencere* (main window) oluşturulabileceğini göreceğiz. Bölüm nihayete erdiğinde bir proğramın grafiksel kullanım arabiriminin tamamını, -*menüler*, *alet çubukları*, *durum çubuğu* ve bütün ihtiyaç duyulan diyaloglar da dahil omlak üzere, oluşturmuş olacaksınız.



Şekil 3.1: Elektronik çizelge proğramı

Ana pencere bir proğramın temelini teşkil eder taki kullanıcı arabiriminin geri kalan unsurları bu temel üzerine inşa edilir. Bu bölümde üzerinde çalışacağımız electronik çizelge proğramının ana penceresi şekil 3.1 de teşhir edilmektedir. İkininci bölümde yazdığımız *Bul*, *Hücreye Git* ve *Tasnif* proğramları bu elektronik çizelge proğramında müstamildirler.

Her GUI proğramının arkasında dosya okumak ve yazmak, verilerle işlem yapmak gibi vazifeler ifa eden kaynak kodu mevcuttur. Dördüncü bölümde bu tür kaynak kod kullanarak elektronik çizelge proğramına bir takım kabiliyetler ekleyeceğiz.

QMainWindow dan Altsınıf Oluşturulması

Bir proğramın *ana penceresi* QMainWindow dan *altsınıf* oluşturularak husule getirilir. İkinci bölümde diyaloglar hakkında öğrendiklerimizin çoğu ana pencereler içinde geçerlidirler çünkü QDialog ve QMainWindow sınıflarının ikiside QWidget ın alt sınıflarıdırlar.

Ana pencereler Qt Designer ın içerisinde oluşturulabilirler ancak biz bu bölümde temrin olması açısından kode yazmak suretiyle ana pencere oluşturacağız. Designer kullanarak ana pencere oluşturmayı tercih ediyorsanız bu konuda *Qt Designerin* kullanım klavuzunda "Bir Ana Pencere Proğramı Oluşturulmas" (Creating a Main Window Application) bölümünde detaylı bilgi bulabilirsiniz. *Elektronik çizelge* proüramının ana penceresinin kaynak kodu mainwindow.h ve mainwindow.cpp dosyaları arasında dağıtılmışdır. Gelin başlık dosyası ile başlayalım:

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <qmainwindow.h>
#include <qstringlist.h>
class QAction;
class QLabel;
class FindDialog;
class Spreadsheet;
class MainWindow : public QMainWindow
{
      Q_OBJECT
public:
      MainWindow(QWidget *parent = 0, const char *name = 0);

protected:
      void closeEvent(QCloseEvent *event);
      void contextMenuEvent(QContextMenuEvent *event);
```

MainWindow sınıfını QMainWindow ın alt sınıfı olarak tanımladık. Kendisine has sinyal ve dilimleri olduğu için Q_ OBJECT makrosunu tanımlamak mecburiyetindedir.

Hayali (virtual) bir fonksiyon olan closeEvent(), QWidget tarafından tedarik edilir ve kullanıcı pencereyi kpattığı zaman otomatik olarak çağrılır. Bu fonksiyonu burada yeniden tanımladık taki kullanıcı pencereyi kapatmaya teşebbüs ettiğinde ona değişiklikleri kaydetmek isteyip istemediğini soralım. Aynı zamanda pencere kapanmadan kullanıcı tercih ve ayarlarını dike kaydetmize imkansağlar.

Aynı şekilde contextMenuEvent() fonksiyonu kullanıcı *sağ fare* tuşuna bastığında yada herhangi bir işletim sistemine has menü tuşuna bastığında çağrılır. Bir *açılır menü* oluşturmak için MainWindow sınıfında bu fonksiyonu yeniden tanımladık.

```
private slots:
      void newFile();
      void open();
      bool save();
      bool saveAs();
      void find();
```

```
        void goToCell();
        void sort();
        void about();
```

Bazı, *Dosya|Yeni* ve *Yardim|Proğram Hakkında* gibi, menü seçenekleri `MainWindiw` sınıfının özel dilimleri olarak ifa edildi. Çoğu dilimler koştuktan sonra geriye herhangi bir değer göndermezler (void return type) `save()` ve `saveAs()` müstesna ki bunların geriye gçnderdikleri data `bool` türündendir. Bir dilim bağlı olduğu sinyalin yayınlanması sonucu koşturuldu ise bu durumda geri göndereceği değer ihmal edilir aksi takdirde nirmal bir C++ fonksiyonu gibi tanımına muvafık olarak gerekli değeri geri gönderir.

```
        void updateCellIndicators();
        void spreadsheetModified();
        void openRecentFile(int param);
    private:
        void createActions();
        void createMenus();
        void createToolBars();
        void createStatusBar();
        void readSettings();
        void writeSettings();
        bool maybeSave();
        void loadFile(const QString &fileName);
        void saveFile(const QString &fileName);
        void setCurrentFile(const QString &fileName);
        void updateRecentFileItems();
        QString strippedName(const QString &fullFileName);
```

Ana pencerenin kullanıcı arabirimini destekleyebilmesi için daha birçok hususi dlime ihtiyacı var.

```
        Spreadsheet *spreadsheet;
        FindDialog *findDialog;
        QLabel *locationLabel;
        QLabel *formulaLabel;
        QLabel *modLabel;
        QStringList recentFiles;
        QString curFile;
        QString fileFilters;

        bool modified; enum { MaxRecentFiles = 5 };
        int recentFileIds[MaxRecentFiles];

        QPopupMenu *fileMenu;
        QPopupMenu *editMenu;
        QPopupMenu *selectSubMenu;
        QPopupMenu *toolsMenu;
        QPopupMenu *optionsMenu;
        QPopupMenu *helpMenu;
        QToolBar *fileToolBar;
        QToolBar *editToolBar;
        QAction *newAct;
        QAction *openAct;
        QAction *saveAct;
```

```
        ...
        QAction *aboutAct;
        QAction *aboutQtAct;

};
#endif
```

Hususi dilimlere ilaveten `MainWindiw` sınıfının çok sayıda hususi değişkenleri vardır. Bunların herbiri zamanı geldikçe açıklanacaktır. Şimdi *ana sınıfın* ifasına (implementation) bir göz atalım:

```
#include <qaction.h>
#include <qapplication.h>
#include <qcombobox.h>
#include <qfiledialog.h>
#include <qlabel.h>
#include <qlineedit.h>
#include <qmenubar.h>
#include <qmessagebox.h>
#include <qpopupmenu.h>
#include <qsettings.h>
#include <qstatusbar.h>

#include "cell.h"
#include "finddialog.h"
#include "gotocelldialog.h"
#include "mainwindow.h"
#include "sortdialog.h"
#include "spreadsheet.h"
```

Qt ye ait başlık dosyalarını eklediğimiz gibi, `finddialog.h`, `gotocelldialog.h`, ve `sortdialog.h` benzeri hususi sınıflarında başlık dosyalarını ekledik.

```
MainWindow::MainWindow(QWidget *parent, const char *name)
        : QMainWindow(parent, name)
{
        spreadsheet = new Spreadsheet(this);
        setCentralWidget(spreadsheet);

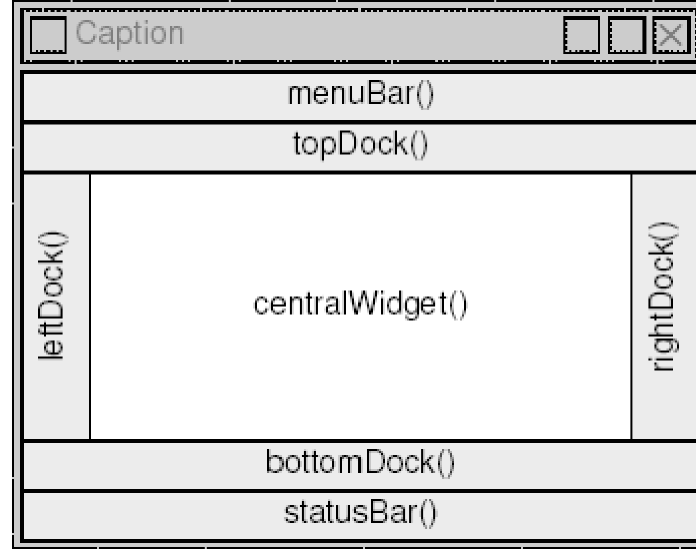
        createActions();
        createMenus();
        createToolBars();
        createStatusBar();

        readSettings();

        setCaption(trUtf8("Elekronik Çizelge"));
        setIcon(QPixmap::fromMimeSource("icon.png"));

        findDialog = 0;
        fileFilters=trUtf8("Elektronik Çizelge Dosyalar  (*.sp)");
        modified = false;
}
```

Yapıcıda (constructor) evvela bir `Spreadsheet` aleti oluşturuyoruz ve onu ana pencerenin *merkezi aleti* tayin ediyoruz. Merkezi alet alet çubukları ile durum çubuğu arasında kalan alanı kaplar. `Spreadsheet` sınıfı `QTable` sınıfının bir alt sınıfı olup elektronik çizelge proğramına has bir takım yetenekler içerir. Bu sınıfın detaylarını dördüncü bölümde göreceğiz.



Şekil 3.1: QMainWindow un ihtiva ettiği aletler.

Daha sonra birer hususi fonksiyon olan `createActions()`, `createMenus()`, `createToolBars()` ve `createStatusBar()` fonksiyonlarını çağırmak suretiyle ana pencerenin geri kalan kısımlarını oluşturuyoruz.

Yine `readSettings()` fonksiyonunu çağırmak suretiyle kullanıcı tercihlerini yüklüyoruz. Pencerenin *simgesi* (ikonu) olarak *PNG* formatındakı icon.png dosyasını kullanıyoruz. Qt, *BMP, GIF*[1], *JPEG, MNG, PNG, PNM, XBM,* ve *XPM* dahil olmak üzere çok sayıda *resim* (imge) formatını tanımaktadır. `QWidget::setIcon()` fonksiyonunu çağırmak suretiyle pencerenin sol üst köşesinde görüntülenecek olan simge belirlenmiş olur. Malesef işletim sisteminden bağımsız masaüstünde görüntelenecek olan simgeyi ayarlamak şu aşamada mümkün değildir. Bu konuda detaylar Qt tarafından şu internet sayfasında tedarik edilmektedir: *http://doc.trolltech.com/3.2/appicon.html.*

GUI proğramları genellikle çok sayıda resim kullanırlar ve bazan aynı resim farklı yerlerden bir kaç defa kullanılabilir. Qt resimleri proğrama bir kaç metod ile tedarik eder. :

- Resimler dosyalarda saklanır ve proğramın çalışması esnasında bu dosyaları yüklenirler.

---

[1] Proğram patentlerini tanıyan ülkelerde Unisys tarafından patentlenen LZD açma (decompression) algorştmasının telif hakkından dolayı Qt normalde GIF destegini sağlamaz. Telif hakkını ödemiş iseniz yada ödemek zorunda değil iseniz Qt deki mevcut GİF destğini açabilirsiniz.

- *XPM* dosyalrı kaynak dosyalarına ilave edilir. (Bunun mümkün olmasının sebebi *XPM* dosyalarının aynı zamanda muteber C++ dosyaları olmalarıdır.)
- Qt de mevcut olan *resim kolleksiyonu mekanizması* kullanılarak.

Burada biz *resim kolleksiyonu mekanizmasıni* kolay, rsimleri proğram çalışırken yüklemekten daha elverişli olduğu ve Qt nin desteklediği bütün resim formatlarının kullanılmasına imkan sağladığı için diğerlerine tercih ediyoruz. Resimler kaynak dodunun muhafaza edildiği dizi altında images namında bir dizi altında saklanırlar.

Proje dosyası olan `.pro` dosyasını aşapıdaki satırları eklemek suretiyle *uic* proğramının bu resimleri ihtiva eden C++ kaynak kodu dosyası oluşturmasını sağlamış oluruz.

```
IMAGES = images/icon.png \
         images/new.png \
         images/open.png \
         ...
         images/find.png \
         images/gotocell.png
```

Bu dosyalar derleyici tarafından derlenip proğramın parçası haline gelirler ve `Qpixmap::fromMimeSource()` fonksiyonunu kullanrak bu resimlere proğram içerisinden ulaşılabilir. Bu yaklaşım *simge* ve *diğer* resimlerin kaybolma tehlikesini ortadan kaldırır.

Qt designer diyalog ve formaların görsel olarak tasarımına izin verdiği gibi *resim kolleksiyonuna* yeni resimler ekleme ımkanınıda sağlar.

Menülerin ve Alet Çubuklarının Oluiturulması

Modern GUI proğrmlarının çoğu hem menü hemde alet çubuğu kullanırlar ve aşağı yukarı aynı komutlar her ikisindede mevcuttur. Menüler kullanıcının proğrama tam hakimiyetine imkan sağlarken alet çubukları sık kullanılan komutlara seri ulaşmasını mümkğn kılarlar. Qt menü ve alet çubuğu yapımını *fiil* (action) diye adladırılan bir mefhum ile kolaylaştırır. *Fiil* bir *ferd* olup ya bir menüye yada bir *alet çubuğuna* veyahutta her ikisine eklenebilir. Qt altında menülerin ve alet kutularının teşkili şu adımlardan ibarettir:

- Fiili (action) oluştur.
- Fiili menüye ekle.
- Fiili alet çubuğuna ekle.

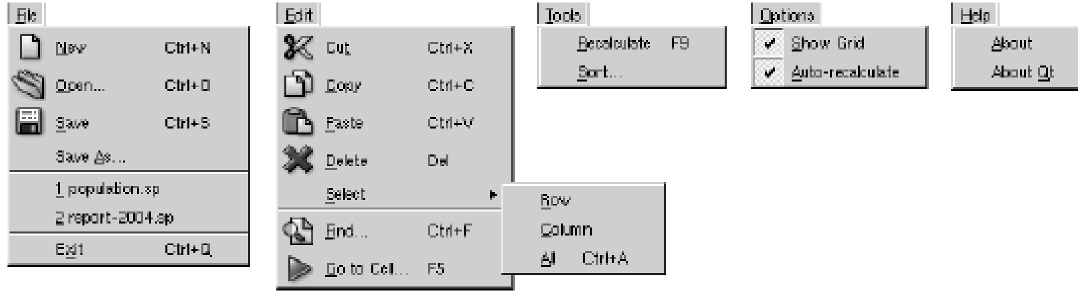
Spreadsheet proğramında *fiiller* (actions) `createActions()` içerisinde meydana getirilirler:

```
void MainWindow::createActions()
{
      newAct=new QAction(trUtf8("&Yeni"), rUtf8("Ctrl+Y"),this);
      newAct->setIconSet(QPixmap::fromMimeSource("new.png"));
      newAct->setStatusTip(trUtf8("Yeni electronik çizelge
                                   pro ram  olu tur"));
      connect(newAct, SIGNAL(activated()), this, SLOT(newFile()));
```

Yeni *fiilinin* (action) bir *kestirme yol tuşu* (Yeni), bir *hızlandırılmış tuşu* (Ctrl+N), bir *atası*

(ana pencere), bir *simgesi* (new.png), ve birde *durum ipucu* vardır. *Fiilin* `activated()` sinyalini ana pencerenin *hususi* (`private`) `newFile()` dilimine, ki onu biz bir sonraki bölümde oluşturacağız, bağlıyoruz. Bu bağlantı olmaksızın kullanıcı her ne kadar *Dosya*| *Yeni* düğmesine veya *Yeni* alet kutusu düğmesine tıklarsa tıklasın hiçbir şey vuku bulmaz.

Dosya, Düzenle ve *Aletler* menülerinin diğer *fiilleri* (eylemleri) *Yeni* eylemine benzerler.



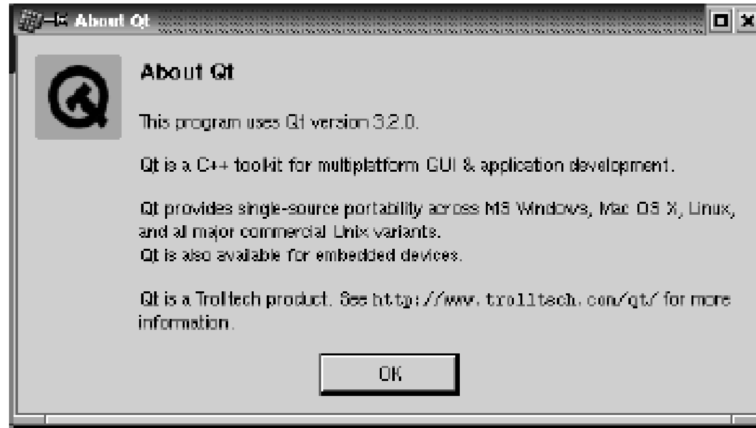
Şekil 3.3: Elektronük çizelge proğramının menüleri

Seçenekler menüsündeki *Izgarayı Göster* fiili farklıdır:

```
showGridAct = new QAction(trUtf8("&Izgaray  Göster"), 0,
                                                       this);
showGridAct->setToggleAction(true);
showGridAct->setOn(spreadsheet->showGrid());
showGridAct->setStatusTip(trUtf8("Elektronik çixelgenin"
                                 " zgaras n   göster/sakla."));
connect(showGridAct, SIGNAL(toggled(bool)), spreadsheet,
                                 SLOT(setShowGrid(bool)));
```

Izgarayı Göster, sakin bir fiildir (toggle action). *Sakin fiil* menü altında onay işareti ile gösterilirken alet çubuğunda *sakin düğme* (toggle button) ile gösterilir. Bu fiil açıldığında izgara görüntülenir. *Izgarayı Göster* ffilinin ilk halini Spreatsheet in varsayılan haline eşitliyoruz taki menü ve alet çubuğu birbiri ile muvafık olsunlar. Sonra bu fiilin `toggled(bool)` sinyalini Spreadsheet in `setShowGrid(bool)` dilimi ile bağlıyoruz, ki bu dilim ona `QTable` dan miras kalmıştır. Bu fiil bir menü yada alet çubuğuna eklendiğinde kullanıcı ızgarayı açıp kapatabilir.

Izgarayı Göster ve *Otomatik Hesapla* sakin fiilleri birbirinden farklı fiillerdir. `QAction`, yanlızca bir üyesinin kullanılmasına izin veren fiil gurubu sınıfı olan `QActionGroup` ilede beraber kullanılır.



Şekil 3.4: Qt Hakkında

```
        aboutQtAct = new Qaction(trUtf8("&Qt Hakk nda"), 0, this);
        aboutQtAct->setStatusTip(trUtf8("Qt Hakk nda diyalo unu"
                                        " göster."));
        connect(aboutQtAct, SIGNAL(activated()), qApp,
                                        SLOT(aboutQt()));
}
```

Qt Hakkında diyaloğunu görüntülemek için `QApplication` nesnesinin `aboutQt()` dilimini kullanıyoruz. Bu dilime küresel bir değişken olan `qApp` vasıtasıyla ulaşabiliriz.

Fiil veya eylemleri oluşturduk şimdi bu fiillerin ifa edilenilmesi için menüleri oluşturabiliriz:

```
void MainWindow::createMenus()
{
        fileMenu = new QPopupMenu(this);
        newAct->addTo(fileMenu);
        openAct->addTo(fileMenu);
        saveAct->addTo(fileMenu);
        saveAsAct->addTo(fileMenu);
        fileMenu->insertSeparator();
        exitAct->addTo(fileMenu);
        for (int i = 0; i < MaxRecentFiles; ++i)
            recentFileIds[i] = -1;
```

Qt de bütün menüler `QPopupMenu` sınıfından neşet ederler. Dosya menüsünü oluşturuyoruz ve daha sonra *Yeni, Aç, Kaydet, Yeni İsimle Kaydet* ve *Çık* afalini (eylemlerini) ona ekliyoruz. Ayırıcılar eklemek suretiyle bir menü altında mütaallik efradı bir araya topluyoruz. Sonra bir `for` döngüsü kullanarak yakın geçmişte açılan dosyalar listesini `recentFilesIds` vektörüne (array) kaydeder. Bir sonraki kısımda *Dosya* menüsünün dilimlerini oluşturuken `recentFilesIds` vektörünü kullanacağız.

```
        editMenu = new QPopupMenu(this);
        cutAct->addTo(editMenu);
        copyAct->addTo(editMenu);
        pasteAct->addTo(editMenu);
        deleteAct->addTo(editMenu);
```

```
        selectSubMenu = new QPopupMenu(this);
        selectRowAct->addTo(selectSubMenu);
        selectColumnAct->addTo(selectSubMenu);
        selectAllAct->addTo(selectSubMenu);
        editMenu->insertItem(trUtf8("&Seç"), selectSubMenu);

        editMenu->insertSeparator();
        findAct->addTo(editMenu);
        goToCellAct->addTo(editMenu);
```

Düzenle menüsünün bi *altmenüsü* mevcuttur. *Altmenü*, sahibi olan *Düzenle* menüsü gibi, bir `QPopupMenu` dür. Altmenüyü this nesnesi atası olacak şekilde oluşturduktan sonra *Düzenle* menüsündeki yerini koyuyoruz.

```
        toolsMenu = new QPopupMenu(this);
        recalculateAct->addTo(toolsMenu);
        sortAct->addTo(toolsMenu);

        optionsMenu = new QPopupMenu(this);
        showGridAct->addTo(optionsMenu);
        autoRecalcAct->addTo(optionsMenu);

        helpMenu = new QPopupMenu(this);
        aboutAct->addTo(helpMenu);
        aboutQtAct->addTo(helpMenu);

        menuBar()->insertItem(trUtf8("&Dosya"), fileMenu);
        menuBar()->insertItem(trUtf8("&Düzenle"), editMenu);
        menuBar()->insertItem(trUtf8("&Aletler"), toolsMenu);
        menuBar()->insertItem(trUtf8("&Seçenekler"), optionsMenu);
        menuBar()->insertSeparator();
        menuBar()->insertItem(trUtf8("&Yard m"), helpMenu);
    }
```

Aletler, *Seçenekler* ve *Yardım* menülerini aynı şekilde oluşturduktan sonra *menü çubuğuna* ekliyoruz. *QMainWindow::menuBar()* fonksüyonu `QmenuBar` için bir *timsal* (pointer) getirir. (`menuBar()` ilk çağrıldığında *menü çubuğu* oluşturulur.) *Seçenekler* menüsüs ile *Yardım* menüsü arasına bir *aralıkçı* yerkeştiriyoruz. *Motif* ve benzeri *sitillerde* bu aralıkçı *Yardım* menüsünü en sağ köşeye iterken diğer *sitillerde* bu aralıkçı ihmal edilir.



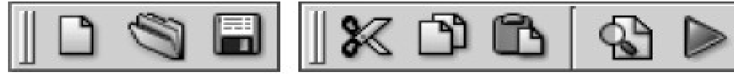
Şekil 3.5: Motif ve Windows tipi menü çubuğu.

Alet çubuğunun oluşturulması menülere çok benzer:

```
    void MainWindow::createToolBars()
```

```
{
      fileToolBar = new QToolBar(trUtf8("Dosya"), this);
      newAct->addTo(fileToolBar); openAct->addTo(fileToolBar);
      saveAct->addTo(fileToolBar);
      editToolBar = new QToolBar(trUtf8("Düzenle"), this);
      cutAct->addTo(editToolBar);
      copyAct->addTo(editToolBar);
      pasteAct->addTo(editToolBar);
      editToolBar->addSeparator();
      findAct->addTo(editToolBar);
      goToCellAct->addTo(editToolBar);
}
```

Doay ve *Düzenle* alet çubuklarının oluşturuyoruz. Menülerde olduğu gibi alet çubuklarıda *aralıkçılar* ihtiva edebilirler.



Şekil 3.6: Elektronşk çizelgesinin alet çubuğu.

Menüleri ve alet çubuklarını tamaladık , artık *siyak* (context) menülerini oluşturabiliriz. Böylece kullanıcı arabirimini tamalamış olacağız:

```
void MainWindow::contextMenuEvent(QContextMenuEvent *event)
{
      QPopupMenu contextMenu(this);
      cutAct->addTo(&contextMenu);
      copyAct->addTo(&contextMenu);
      pasteAct->addTo(&contextMenu);
      contextMenu.exec(event->globalPos());
}
```

Kullanıcı sağ fare tuşuna bastığında (veya bazı kılaviyelerde menü tuşuna bastığında) *siyak* menüsü vakasının (olayının) varlığından ilgili alet haberdar edilir. `QWidget::contextMenuEvent()` fonksiyonunu yeniden tanımlamak suretiyle *siyak* menüsünü kullanıcının fare tuşuna bastığı noktada görüntüleyebiliriz.



Şekil 3.7: Elektronşk çizelgesinin *siyak* menüsü.

Sinyaller ve *dilimler* gibi vakalrda Qt nin temel unsurlarındandır. *Vakalar* (events) Qt nin çekirdeği tarafından fare tıklamalrı, tuşa basılması, pencerelerin ebatatlarının değiştirilmesi hadisesi ve benzeri hadiseleri ihbar etmek için husule getirilirler. Onlar *hayali* (virtual) fonksiyonları yeniden tanımlamak suretiyle gerçekleştirilebilirle.

Siyak menüsüsnü `MainWindow` içerisinde oluşturduk çünkü *eylemler* burada muhafaza

edilirler. Siyak menüsünü Spreadsheet sınıfı içerisinde de oluütuemak mümkündür. Kullanıcı elektronik çizelge proğramının aletlerinden birirsinin üzerinde sağ fare tuşuna tıkladığında Qt önce o alete *siyak menü vakası* gönderir. Şayet Spreadsheet sınınfı `contextMenuEvent()` fonksüyonunu yeniden tanımlamış ise, vaka orada son bulur aksi takdirde o sözkonusu aletin atasına (`MainWindow`) gönderilir. Vakalar 7. bölümde detayları ile birlikte ele alınacaklardır.

Siyak menü vakasına halleden kode şu ana kadar karıılaştıklarımızdan farklıdır çünkü o *yığında* (stack) değişken olan bir alet (`QPopupMenu`) oluşturur. `New` ve `delete` de kullanılabilirdi burada:

```
QPopupMenu *contextMenu = new QpopupMenu(this);
cutAct->addTo(contextMenu);
copyAct->addTo(contextMenu);
pasteAct->addTo(contextMenu);
contextMenu->exec(event->globalPos());
delete contextMenu;
```

Bir diğer üzerinde durulması gereken nokta `exec()` fonksüyonudur. `QPopupMenu::exec()` `sirak menüsünü` ekranın belirli bir yerinde görüntüler ve kullanıcı bir seçeneği ihtiyar edince veya menüyü kapatınca geri döner. Bu aşamada `QPopupMenu` *nesnesi* vazifesini ifa etmiş olur ve artık onu *imha* edebiliriz. Şayet `QPopupMenu` *nesnesi yığında* yer alıyorsa fonksiyonun sonunda otomatik olarak *imha* edilir aksi takdirde `delete` çağırmak suretiyle onu yok etmemiz lazım.

Böylece menüler ve alet çubuklarının kullanıcı arabirimi kısmını tamamlamış olduk. Ancak dilimlerin tamamını yada yakın geçmişte açılmış olan dosyaları takip edecek olan kaynak kodunu henüz yazmadık. Gelecek kısımda bu hususları halledeceğiz.

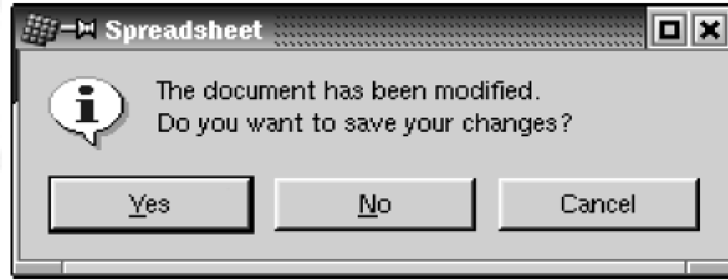
Dosya Menüsünün İfası

Bu kısımda *Dosya* menüsünün çalıiması için gerekli olan dilim ve hususi fonksiyonları bu kısımda geliştireceğiz.

```
void MainWindow::newFile()
{
      if (maybeSave())
      {
            spreadsheet->clear();
            setCurrentFile("");
      }
}
```

Kullanıcı *Dosya*|*Yeni* menüsüsnü seçtiğinde yada *Yeni* alet çubuğu düğmesine bastığında `newFile()` dilimi çağrılır. `maybeSave()` hususi fonksiyonu kullanıcıya varsa yaptığı değişiklikleri diske kaydetmek isteyip istemediğini sorar. Eğer kullanıcı `Evet` düğmesine basarsa bu fonksiyon *müsbet* aksi takdirde (`Vazgeç` veya `Hay r` tuşuna basarsa) *menfi* geri getirir. `setCurrentFile()` hususi fonksiyonu pencerenin başlığını değiştirerek *isimsiz* bir

dosyanın edit edilmekte olduğunu gösterir.



Şekil 3.8: Elektronşk çizelgesinin *siyak* menüsü.

```
bool MainWindow::maybeSave()
{
    if (modified)
    {
        int ret=QMessageBox::warning(this, tr("Spreadsheet"),
        trUtf8("Dosya de i tirildi.\n"
        "Kaydetmek istermisin?"),
        Qobject::trUtf8("Evet"),
        Qobject::trUtf8("Hay r"),
        QObject::trUtf8("Vazgeç"),
        QMessageBox::Yes | QMessageBox::Default,
        QMessageBox::Cancel | QMessageBox::Escape);
        if (ret == QMessageBox::Yes)
            return save();
        else if (ret == QMessageBox::Cancel)
            return false;
    }
    return true;
}
```

`maybeSave()` fonksiyonu içerisinde Şekil 3.8 de izhar edilen risale kutusunu (mesaj diyaloğunu) oluşturuyoruz. Bu mesaj kutusunun *Evet*, *Hayır* ve *Vazgeç* adında üç düğmesi mevcuttur. `QMessageBox::Default` eklemek suretiyle *Evet* düğmesini varsayılan düğme yaptık. `QmessageBox::Escape`, *Esc* tuşunu *Yok* un müradifi yapar, yani *Esc* tuşuna basmak *Yok* tuşuna basmakla aynı manaya gelir.

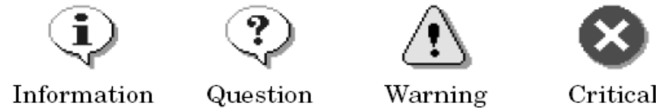
`warning()` fonksüyonu ilk etapta karmaşık gibi görünsede genel anlamda *nehvi* (syntax) gayet açıktır:

```
QMessageBox::warning(parent[2], caption[3], messageText[4], button0[5],
```

[2] ata
[3] başlık
[4] izhar edilmesi arzu edilen metin
[5] ilk düğme

```
button1[6], ...);
```

`QMessageBox` sınıfı aynı zamanda `information()`, `question()`, ve `critical()` fonksiyonlarınıda ihtiva eder, kı onlar `warning()` fonksiyonu gibi davranırlar ancak farklı *simge* (icon) görüntülerler.



Şekil 3.9: Mesaj kutusu simgeleri[7].

```
void MainWindow::open()
{
    if (maybeSave())
    {
         QString fileName =
            QFileDialog::getOpenFileName(".",fileFilters,this);
         if (!fileName.isEmpty())
            loadFile(fileName);
    }
}
```

`open()` dilimi *Dosya*|*Aç* menüsüne tekabül ve taalluk eder. `newFile()` gibi önce kaydedilmemiş olan değişiklikleri halletmek için `maybeSave()` fonksiyonunu çağırır. Sonra statik bir fonksiyon olan `QFileDialog::getOpenFileName()` çağırmak suretiyle dosya ismini elde eder. Bu fonksiyon dosya diyaloğunu açar ve kullanıcının bir dosya seçmesine imkan sağlar. Şayet bir dosya sçilmiş ise dosya ismini geri getirir yok eğer kullanıcı *Vazgeç* (cancel) tuşuna basar ise o durumda boş bir metin geri getirir.

`getOpenFileName()` fonksiyonunu üç değişken ile çağırıyouz. İlk değişken, hangi *diziden* başlması gerektiğini belirtir. Yukarıdaki misalde ".", *hali hazırdaki diziden*[8] başlamasi anlamına gelir. İkinci değişken, `fileFilters`, dosya süzgecini yada filitresini ihtiva eder. Bir filitre açıklama ve genel arama karakterinden (wildcard) ibarettir. `MainWindow` *yapıcısında* (constructor) , `fileFilters` şu şekilde oluşturulmuştu:

```
fileFilters = tr("Elektronik Çizelge Dosyalar  (*.sp)");
```

Elektronik çizelge proğramının kendine mahsus dosya uzantısına ilaveten başka tür dosyalarıda virgül ile yek diğerinden ayrılmiş halde şu şekilde belirtebilirdik:

---

[6] ikinci düğme

[7] `information()`, `question()`, `warning()` ve `critical()` fonksüyonları sırasıyla *informasyon*, *soru*, *uyarı* ve *hata* görüntülemek için kullanılırlar.

[8] Unix/Linux altında proğramın başlatıldığı dizi

```
fileFilters = trUtf8("Elektronik Çizelge Dosyalar  (*.sp)\n"
      "Virgül ile ayr lmi  de erler (*.csv)\n"
      "Lotus 1-2-3 dosyalar  (*.wk?)");
```

`getOpenFileName()` fonksiyonunun son argümanı olan değişken bu diyaloğun atasını belirlerki yukarıdaki misalde dosya diyaloğu ana pencerenin çocuğu olmuş olur. *Ata* ve *çocuk* arasındaki ilişki, *diyaloglar* için, *aletlerden* biraz farklıdır. *Diyalog* başlı başına ön pilanda yer alan bir alet olup atası versa onun önünde ve ortasında yer alır. *Diyalog* atasının `durum çubu u` girdisini *atası* ile paylaşır.

```
void MainWindow::loadFile(const QString &fileName)
{
   if (spreadsheet->readFile(fileName))
   {
     setCurrentFile(fileName);
     statusBar()->message(trUtf8("Dosya yüklendi"),2000);
   }
   else
   {
      statusBar()->message(trUtf8("Yüklemeden vazgeçildi"),2000);
   }
}
```

`loadFile()` hususi fonksiyonu dosya yüklemek için `open()` içerisinde çağrılmaktadır. Onu bağımsız fonksiyon olarak oluşturduk çünkü daha sonra onu kullanarak yakın geömişte aöılmış dosyaları yükleyeceüiz.

`Spreadsheet::readFile()` fonksiyonunu kullanarak sodyayı diskten yüklüyoruz ve şayet yükleme işlemi başarı ile tamamlando ise `setCurrentFile()` fonksiyonunu kullanarak pencerenin başlığını ayarlıyoruz. Aksi takdirde `Spreadsheet::loadFile()` fonksiyonu kullanıcıyı dosya açarken problem ortaya çıktı diye bir mesaj kutusu ile haberdar eder. Genelde hata meydana geldiğinde ona en yakın olan birim tarafından ihbar edilmesi tercih edilir çünkü muhbir hataya yakınlığı nisbetinde onun hakkında detaylara saipdir.

Behemehal durum çubuğunda 2000 mili saniyeliğine (2 saniye) kullanıcıyı olup bitenlerden haberdar etmek için mesaj görüntülüyoruz.

```
bool MainWindow::save()
{
      if (curFile.isEmpty())
      {
            return saveAs();
      }
      else
      {
            saveFile(curFile);
            return true;
      }
}
```

```
void MainWindow::saveFile(const QString &fileName)
{
     if (spreadsheet->writeFile(fileName))
     {
          setCurrentFile(fileName);
          statusBar()->message(tr("File saved"), 2000);
     }
     else
     {
          statusBar()->message(tr("Saving canceled"), 2000);
     }
}
```

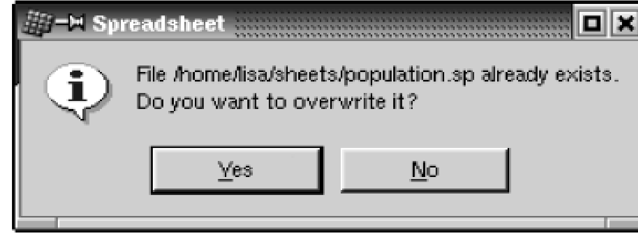
`save()` dilimi *Dosya|Kaydet* menüsüne tekabül eder. Dosyanın bir ismi var ise yada daha önce kaydedilmiş ise `saveFile()` aksi takdirde `saveAs()` fonksiyonunu çağırır.

```
bool MainWindow::saveAs()
{
    QString fileName = QFileDialog::getSaveFileName(".",
                                             fileFilters, this);
    if (fileName.isEmpty())
       return false;
    if (QFile::exists(fileName))
    {
       int ret = QMessageBox::warning(this, trUtf8("Spreadsheet"),

            trUtf8("%1 dosyas  zaten mevcut.\n"
               "Üzerine yazmak istermisiniz?")
            .arg(QDir::convertSeparators(fileName)),

            Qobject::trUtf8("Evet"),
            Qobject::trUtf8("Hay r"),
            NULL,
            QMessageBox::Yes | QMessageBox::Default,
            QMessageBox::No | QMessageBox::Escape);
       if (ret == QMessageBox::No)
            return true;
    }
    if (!fileName.isEmpty())
      saveFile(fileName);
      return true;
}
```

`saveAs()` dilimi *Dosya|Yeni İsimle Kaydet* menüsüne tekabül eder. `QFileDialog::getSaveFileName()` fonksiyonunu çağırmak suretiyle kullanıcıdan dosya ismini taleb ediyoruz. Kullanıcı V*azgeç* (Cancel) tuşuna basarsa o zaman *menfi* (false) geri dönderip bunu ta `maybeSave()` fonksiyonuna kadar ulaştırıyoruz. Aksi takdirde var olan veya yeni bir dosya ismini geri getiriyoruz. Bu isimde bir dosya mevcut ise, `QMessageBox::warning()` çağrılarak Şekil 3.10 teşhir edilen mesaj kutusunu izhar edilir ve kullanıcı uyarılır.



Şekil 3.10: Mevcut dosyanın üzerine yazmak istermisin?

Mesaj kutusuna gönderilen metin şudur:

```
trUtf8("%1 dosyas  zaten mevcut.\n"
"Üzerine yazmak istermisiniz?")
.arg(QDir::convertSeparators(fileName))
```

`QString::arg()` fonksiyonu en küçük numaralı "%n" parametresini kendi argumanı (burada fileName) ile değiştirip husule gelen metni geri gönderir. Mesela, dosya ismi A:\tab04.sp ise, yukarıdaki kod,proğramın bir baika dile çevrilmediğini varsayarsak, şuna eşdeğerdir:

```
" A:\tab04.sp dosyas  zaten mevcut.\n"
"Üzerine yazmak istermisiniz?"
```

`QDir::convertSeparators()` fonksiyonu Qt tarafından dizi ayırıcısı olarak kullanılan '/' karekterini işletim sistemine uygun dizi ayırıcısına öeviri (Unix ve Mac OS X işletim sistemi için '/ ' ve Windows işletim sistemi için '\').

```
void MainWindow::closeEvent(QCloseEvent *event)
{
      if (maybeSave())
      {
            writeSettings();
            event->accept();
      }
      else
      {
            event->ignore();
      }
}
```

Kullanıcı *Dosya|Çık* menüsünü seçtiğinde yada pencerenin başlık kısmındaki X e tıkladığında `QWidget::close()` dilimi çağrılır. Bu alete kapan eylemi gönderir. `Qwidget::closeEvent()` fonksiyonunu yeniden tanımlamak suretiyle, kullanıcının pencereyi kapatma eylemini karşılayabilir ve pencerenin kapatılması gerekip gerekmediğine kendimiz karar verebiliriz.

Eğer kaydedilmemiş değişiklikler var ise ve kullanıcı *Vazgeç* tuşuna basmış ise bu durumda pencere kapanma eylemini "ihmal" ederiz. Aksi takdirde pencere kapatma eylemini kabul ederiz buda proğramın sona ermesine nedne olur.

```
void MainWindow::setCurrentFile(const QString &fileName)
```

```
{
      curFile = fileName;
      modLabel->clear();
      modified = false;
      if (curFile.isEmpty())
      {
            setCaption(trUtf8("Elektronik çizelge"));
      }
      else
      {
            setCaption(trUtf8("%1 - %2")
            .arg(strippedName(curFile))
            .arg(trUtf8("Elektronik çizelge")));
            recentFiles.remove(curFile);
            recentFiles.push_front(curFile);
            updateRecentFileItems();
      }
}

QString MainWindow::strippedName(const QString &fullFileName)
{
      return QFileInfo(fullFileName).fileName();
}
```

`setCurrentFile()` fonksiyonu içerisinde edit edilmekte olan dosya ismini içeren `curFile` *hususi değişkenini* ayarlıyoruz ve dosyada değişiklik yapılıp yapılmadığını gösteren DEĞ işaretini *durum çubuğundan* siliyoruz. `arg()` fonksiyonunun iki tane %n paremetreleri ile çağrıldığına dikkate et. `arg()` fonksiyonuna yapılan ilk çağrı "%1" i değiştirirken; ikinci çağrı "%2" i değiştirir. Bunun yerine

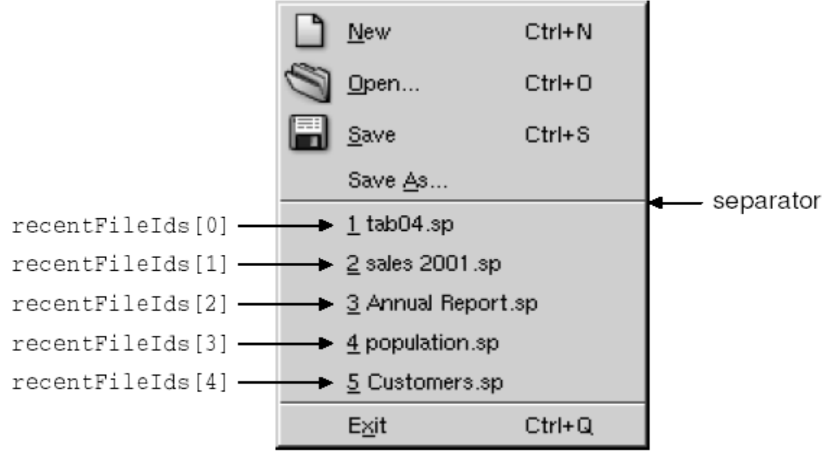
```
setCaption(strippedName(curFile) + tr(" - Elektronik çizelge"));
```

yazmak daha kolay olurdu ancak `arg()` fonksiyonunun kullanılması proğramın başka dillere çevrilmesini elverişli kılar. Doayanın *yolunu* (path) `strippedName()` fonksiyonunu kullanarak çıkararak yanlız dosya ismini sadelik için muhafaza ediyoruz.

Dosya ismi var ise bu durumda geçmişte açilmiş doay listesini ihtiva eden `recentFiles` değişkenini güncelleştiriyoruz. `remove()` fonksiyonunu kullanarak bu dosya ismini mevcut ise mezkur listeden siliyoruz sonra `push_front()` fonksiyonunu kullanarak dosyayı listenin başina ekliyoruz. Ayni dosyanın listede birden fazla defa bulunmasını önlemek için `remove()` fonksiyonunu çağırmak zaruridir.

Listeyi güncelleştirdikten sonra `updateRecentFileItems()` fonksiyonunu kullanarak Dosya menüsündeki geçmişte açılan dosya isimleri listesini hazırlarız. `recentFiles` depişkeni `QStringList (list of Qstrings)` sınıfının bir nesnesidir. Bölüm 11 de Qt de mevcut olan `muhtevi` sınıflarını detaylı olarak göreceğiz ve bunların C++ STL kütüphanesi (Standard Template Library) ile ilişkileri hakkında bilgi edineceğiz.

Böylece dosya menüsünü nerde ise tamamladık. Geriye sadece bir fonksiyon ve istinad amaçlı bir *dilim* kaldi. Her ikiside son zamanda açılmış olan dosyalar listesi ile alakalıdırlar.



Şekil 3.11: Dosya menüsü.

```
void MainWindow::updateRecentFileItems()
{
      while ((int)recentFiles.size() > MaxRecentFiles)
            recentFiles.pop_back();
      for (int i = 0; i < (int)recentFiles.size(); ++i)
      {
            QString text = trUtf8("&%1 %2")
                  .arg(i + 1)
                  .arg(strippedName(recentFiles[i]));
            if (recentFileIds[i] == -1)
            {
               if (i == 0)
                  fileMenu->insertSeparator(fileMenu->count()-2);
               recentFileIds[i] =
                              fileMenu->insertItem(text,this,
                              SLOT(openRecentFile(int)),
                              0, -1,
                              fileMenu->count() - 2);
               fileMenu->setItemParameter(recentFileIds[i],i);
            }
            else
            {
               fileMenu->changeItem(recentFileIds[i], text);
            }
      }
}
```

`updateRecentFileItems()` hususi fonksiyonunu kullanarak yakın geçmişte açilmiş olan dosya listesini hazırlıyoruz. Önce `recentFiles` listesinde izin verilende (`MaxRecentFiles`

değişkeni in `mainwindow.h` dosyasında beş ile sınırlanmıştır) fazla dosya bulunmaması için varsa ziyade olanları listenin sonundan siliyoruz.

Sonra bu listedeki her bir dosya için bir menü ferdi oluşturuyoruz yada mevcut olan ferdi yeniden kullanıyoruz. Menü ferdini ilk defa oluşturduğumuzda buna ilaveten birde ayırıcı oluşturuyoruz. Bu işlem `createMenus()` yerine burada yağıyoruz taki birden fazla ayırıcının ard arda kullanılmasına engel olmuş olalım. `setItemParameter()` fonksiyonuna yapılan çağrıyı birazdan açıklayacağız.

Bir takım ferdleri `updateRecentFileItems()` içerisinde oluşturmamız ve onları silememiz tuhaf gibi görünebilir. Bunun sebebi bir celse (session) esnasında açık dosya listesini büzülmeyeceğini yada küçülmeyeceğini varsayabilmemizdir.

Çağırdığımız `QPopupMenu::insertItem()` fonksiyonunun nehvi şu şekildedir:

```
fileMenu->insertItem(text,receiver,slot,accelerator,id,index);
```

`text` değişkeni menüde ibraz edilen metni ihtiva eder. `strippedName()` fonksiyonunu kullanmak suretiyle dosya isimlerinin içerisinden yol kısmını çıkarıyoruz. Dosya isimlerinin tamamınıda kullanabilirdik ancak bu dosya menüsünün çok geniş olmasına sebep olurdu. Şayet dosya isminin tamamı görüntülenmek istenirse o halde bir alt menü oluşturulması tavsiye ederiz.

`receiver` (alıcı) ve `slot` (dilim) parametreleri kullanıcını bu menü ferdini seçmesi durumunda hangi dilimin çağrılacağını belirler. Bizim misalimizde, `MainWindow` un `openRecentFile(int)` dilimine bağlanıyoruz.

Her bir *hızlandırıcı* (accelerator) ve *kimlik* (id) için varsayılan değerleri kullanıyoruz buda hızlandırıcının olmadığına ve kimlik numarasının (ID) otomatik olarak tayin edildiği anlamına gelir. Tayin edilen ID yi `recentFileIds` listesinde daha sonra ona ulaşabilmemiz için saklıyoruz.

`index` ise ferdi yerleştirmek istediğimiz yeri belirler. `fileMenu->count() - 2` değerini kullanmak suretiyle *Çık* ferdinin ayırıcısının hemen üzerine yerleştiriyoruz.

```
void MainWindow::openRecentFile(int param)
{
      if (maybeSave())
      loadFile(recentFiles[param]);
}
```

Dosya menüsünden daha önce açılöıi olan bir dosya seçildiğinde `openRecentFile()` dilimi çağrılır. Türü `int` olan param parametresi daha çnce `setItemParameter()` ile tayin ettiğimiz değerdir. Bu değerleri seçerken onların daha sonra `recentFiles` listesinin indeks leri olarak kullanılacaklarını göz önünde bulundurduk.

Şekil 3.12: Dosya menüsü.

Menu items

ID	text	param
-32	1 tab04.sp	0
-33	2 sales 2001.sp	1
-34	3 Annual Report.sp	2
-35	4 population.sp	3
-36	5 Customers.sp	4

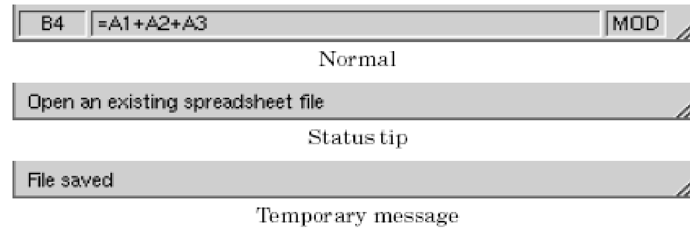
Recently opened files

index	value
0	A:\tab04.sp
1	C:\sales 2001.sp
2	D:\Annual Report.sp
3	C:\population.sp
4	C:\Customers.sp

Bu hali hazırdaki problemi halletmenin bir yoludur. Daha az zarif olan bir yol ise beş tane eylem oluşturup bunlardan her birini ayrı bir dilime bağlamak olurduç

Durum Çubuğunun Oluşturulması

Menü ve *alet çubuklarını* tamamlamakla birlikte şimdi Elektronik çizelge proğramının *durum çubuğunu* oluşturabiliriz. Normalde durum çubuğu üç müşir ihtiva eder: ani kulaanılan hücre, hücrenin ihtiva ettiği formül, ve DEĞ (MOD). Durum çubuğu bunlara ilaveten fani mesajlar ile ip ucu veya yardım mahiyetinde kısa risaleler gösterir.



Şekil 3.13: Dosya menüsü.

`MainWindow` yapıcısı (constructor) `createStatusBar()` çaşırmak suretiyle durum çubuğunu oluşturur:

```
void MainWindow::createStatusBar()
{
        locationLabel = new QLabel(" W999 ", this);
        locationLabel->setAlignment(AlignHCenter);
        locationLabel->setMinimumSize(locationLabel->sizeHint());

        formulaLabel = new Qlabel(this);

        modLabel = new QLabel(trUtf8(" DE  "), this);
        modLabel->setAlignment(AlignHCenter);
        modLabel->setMinimumSize(modLabel->sizeHint());
        modLabel->clear();

        statusBar()->addWidget(locationLabel);
        statusBar()->addWidget(formulaLabel, 1);
        statusBar()->addWidget(modLabel);
```

```
        connect(spreadsheet, SIGNAL(currentChanged(int, int)),
                      this, SLOT(updateCellIndicators()));
        connect(spreadsheet, SIGNAL(modified()),
                      this, SLOT(spreadsheetModified()));
        updateCellIndicators();
    }
```

`QmainWindow::statusBar()` fonksiyonu durum çubuğunun timsalini (pointer) verir. (Durum çubuğu `statusBar()` fonksiyonu ilk çağrıldığında oluşturulur.) Durum çubuğunun ihtiva ettiği müşirler aslında birer etşketlerdir (`QLabels`) ki etiketin metnini biz gerek gördüğümüz zaman değiştiriyoruz. Bu etiketleri (yani `QLabels` nesnelerini) oluştururken onların atası olarak `this` nesnesini kullanıyoruz ancak bunun bir ehemmiyeti yoktur çünkü `QStatusBar::addWidget()` fonksiyonu otomatik olarak durum çubuğunun çucıkları yapar.

Şekil 3.13 de ğörüldüğü üzere bu üç etiketten her birisi farklı büyüklükte bir yere ihtiyaç duyarlar. DEĞ müşiri az bir yer kaplarken pencerenin ebatının değiştirilmesi durumunda ziyade boşluk formülü gösteren müşire tayin edilmelidir. Bunu yapabilmek için formül etiketini eklerken esneme faktörü (stretch factor) olarak `QStatusBar::addWidget()` fonksiyonunu çağırırken 1 kullanıyoruz diğer iki müşir için 0 kullanıyoruz. Esneme faktörünün sıfır olması bu aletin esnemememyi tercih etmesi anlamına gelir. .

`QStatusBar` müşirleri yerleştiriken her bir aletin en makul ebatını `QWidget::sizeHint()` fonksiyonu vasıtası ile çürenir ve esneme kabiliyeti olan aletleri germek suretiyle boşlukları doldurmaya çalışır. Bir aletin ideal veya makul boyutu türüne ve muhteviyatına bağlı olduğu işin o aletin içeriği değiştikçe ideal boyutuda değişir. DEĞ ve hali hazırda kullanılan hğcreyi gösteren müşirlerin daimi bir şekilde ebat değiştirmelerini önlemek için bunların boyutlarını şhtşva edecekleri en uzun metne göre (W999 and DEĞ), biraz boşluk bırakacak şekilde, ayarladık. Onların hizalarını üzerilerindeki metinleri yatay olarak ortalayacak şekilde göstermelri içim `AlignHCenter` olarak ayarladık.

Fonksiyonun sonuna yakın Spreadsheet in iki sinyalini `MainWindow` un iki dilimine bağladık: `updateCellIndicators()` ve `spreadsheetModified()`.

```
void MainWindow::updateCellIndicators()
{
      locationLabel->setText(spreadsheet->currentLocation());
      formulaLabel->setText(" " + spreadsheet->currentFormula());
}
```

`updateCellIndicator()` dilimi hücre pozisyonu ve formülü müşirlerini güncelleştirir. Kullanıcı bir hücreden bir başka hücreye geçtiğinde bu dilim çağrılır. Bu dilim adi bir fonksiyon olarakta `createStatusBar()` fonksiyonunun nihayetindede müşirleri ilk etapta ayarlamak için kullanılır. Bu lüzımludur çünkü `Spreadsheet` ilk başlarken `currentChanged()` sinyalini yayınlamaz.

```
void MainWindow::spreadsheetModified()
{
      modLabel->setText(trUtf8("DE "));
```

```
        modified = true;
        updateCellIndicators();
    }
```

spreadsheetModified() dilimi bütün müşirleri güncelleştirir ve modified (değişti) değişkenini müsbet yapar. (Biz modified değşikenini Dosya menüsünü yaparken kaydedilmeyen değişikliklerin varlığını tesbit etmek için kullandık.)

Diyalogların İstimali

Bu kısımda, diyalogların Qt de nasıl oluşturulabilecekleri, nasıl ayarlanabilecekleri, kullanıcı komutları karşısında nasıl davranmaları gerektiğini göreceğiz. İkinci bölümde oluşturduğumuz Ara, Hücreye Git ve Tasnif diyaloglarını burada kullanacağız. Aynı zmanda bsi bir "Proğram Hakkında" diyaloğu oluşturacağız.

Ara diyaloğu ile başlayacağız. Kullanıcı proğramın ana penceresi ile *Ara* diyaloğu arasında arzu ettipi zaman gidip gelebilmesine imkan sağlamak için Ara diyaloğunu *baki* (modsuz, modeless) pencere olarak oluğturacağız. Baki (modsuz) bir pencere proğramdaki diğer bütün pencerelerdenbağımsız olarak çalişan penceredir. Baki (modsuz) diyaloglar husule getirildiklrinde bunların sinyalleri kullanıcı komutlarına cevap verebilecek olan dilimlere bağlanırlar.

```
    void MainWindow::find()
    {
        if (!findDialog)
        {
            findDialog = new FindDialog(this);
            connect(
              findDialog, SIGNAL(findNext(const QString &, bool)),

              spreadsheet, SLOT(findNext(const QString &, bool)));

            connect(
              findDialog, SIGNAL(findPrev(const QString &,bool)),
              spreadsheet,SLOT(findPrev(const QString &, bool)));
        }
        findDialog->show();
        findDialog->raise();
        findDialog->setActiveWindow();
    }
```

Ara diyaloğu kullanıcının Elektronik çizelhe içerisinde metin aramasını sağlar. Kullanıcı *Düzenle|Ara* menüsünü seçtiğinde find() dilimi çağrılır ve *Ara* diyaloğu görüntülenir. Bu aşamada bir kaç senaryo mümkündür:

- Kullanıcı ilk defa Ara diyaloğunu açtı.
- Ara diyaloğu daha önce açılmış ve kapatılmış.
- Ara diyaloğu daha önce açılmış ve halen daha açık ve meridir yani görülebilir.

Ara diyaloğu mevcut değil ise onu oluşturuyoruz ve daha sonra onun `findNext()` ve `findPrev()` sinyallerini `Spreadsheet` in mukabil dilimkerine bağlıyoruz. İsteseydik diyaloğu `MainWindow` un yapıcısındada oluşturabilirdik ancak bu proğmanın başlama zamanını uzatırdı. Aynı zamanda *Ara* diyaloğu hiç kullanılmaz ise boş yere zaman ve bellek harcamığ oluruz.

Sonra `show()`, `raise()`, ve `setActiveWindow()` fonksiyonlarını çağırmak suretiyle diyaloğun diğer bütün pencerelerin üstüne görülebilir ve faal bir vaziyet almasını sağlıyoruz. Saklı olan yada gözükmeyen bir pencerenin meri olmasını sağlamak için `show()` fonksiyonunu çağırmak yeterlidir ancak *Ara* diyaloğu önceden açılmış ve görünür vaziyettes ise `show()` fonksiyonu hiç bir şey yapmaz. Biz diyaloğun hali hazırdaki durumundan bağımsız olarak onu görüntülemek ve faal yani aktif hale getirmek şstegiğimiz için `raise()` ve `setActiveWindow()` fonksiyonlarını çağırmak zorundayız. Başka bir yaklaıımda

```
if (findDialog->isHidden())
{
      findDialog->show();
}
else
{
      findDialog->raise();
      findDialog->setActiveWindow();
}
```

ieklinde olabilir ki bu hız sınırının 100 km/saat olduğu yerde 90 km/saat hız işe seyretmeye benzer.

Şimdi *Hücreye Git* diyaloğuna bakacağız. Hücreye git diyaloğunun davranışı Ara diyaloğundan biraz farklı olacak. Kullanıcı bu diyaloğu açarö kullanır ve başka bir pencereye gidebilmesi için bu diyaloğu kapatmak zorundadır. Bu da Hücreye Git diyaloğunun *fani* (modlu) bir diyalog olması anlamına gelir. *Fani* (modlu) giyalog açıldığında kullanıcının sözkonusu proğramın herhangi bir penceresine gitmeye izin vermez takı kullanıcı bu pencereyi kapatsın. Ara diyaloğu haricindeki şu ana kadar kullandığımız bütün diyaloglar *fani* (modlu) diyaloglardır.

Bir diyalog `show()` fonksiyonu çağrılarag gösterilmiş ise (`setModal()` fonksiyonunu daha önce çağırmamak şartıyla) modsuzdur, eğer `exec()` fonksiyonu çağrılarak gösterilmiş ise o zaman modlu olur. Modlu diyalogları exec() fonksiyonunu çağırarak oluiturduğumuzda, genelde sinyal dilim bağlantıları yapmamaıza gerek yoktur.

```
void MainWindow::goToCell()
{
      GoToCellDialog dialog(this);
      if (dialog.exec())
      {
            QString str = dialog.lineEdit->text();
            spreadsheet->setCurrentCell(str.mid(1).toInt() - 1,
                                  str[0].upper().unicode() -  A );
      }
```

```
    }
```

`QDialog::exec()` fonksiyonu diyakoğun kabul edilmesi durumunda (*Git* düğmesine basılması yani `accept()`[9] diliminin çağrılması) `müsbet` (true) aksi takdirde `menfi` (false) geri getirir. (Hatırlanırsa ikinci bölğmde Qt Designer ile *Hücreye Git* diyaloğunu oluştururken Git düğmesini `accept()` dilimine ve *Vazgeç* düğmesini ise `reject()`[10] dilimine bağladık.) *Kullanıcı Git* düğmesine basarsa satır muharririndeki hücreyi *aktif* (faal) yapıyoruz, kullanıcı *Vazgeç* düğmesine basarsa `exec()` fonksiyonu menfi geri getirir ve bu durumda yapılacak bir şey yoktur.

The `QTable::setCurrentCell()` fonksiyonu birisi sıra diğeri sütun indeksi olmak üzere iki tane argümant alır. Elektronik çizelge proğramında A1 hücresi (0,0) a tekabül ederken B27 hücresi (26,1) e tekabül eder. `QlineEdit::text()` tarafından geri getirilen `QString` den satır numarasını çıkarmak için QString::mid() fonksiyonunu (bu fonksiyon `QString` türündekü bir değişkenin ihtiva ettiği metni belirli bir indeksten başlayıp sonuna kadar olan parçasını getirir) kullanıyoruz. Daha sonra `Qstring::toInt()` kullanmak suretiyle onu int türüne yani tam sayıya çeviriyoruz ve bu sayıdan bir çıkarmak suretiyle onu sıfır tabanlı ındeks haline dönüştürüyoruz. Sütun numarasını bulmak için kullanıcının vermiş olduğu büyük harfin ASCII değerinden A nın ASCII değerini çıkarıyoruz.

Ara diyaloğunun aksine *Hücreye Git* diyaloğu yığında (stack) oluşturulur. Bu yaklaşım kullanıldıktan sonra ihtıyaş duyulmayan *siyak* menüleri ve *fani* (modlu) diyaloglar GUI azaları için yaygın olarak kullanılır.

Şimdi sıra *Tasnif* diyaloğunda. Tasnif diyaloğu fani bir diyalog olup seçilmiş alanın kullanıcının belirlemiş olduğu bir sütuna göre tasnif eder. Şekil 3.14 bir tasnif misali ihtiva ederki orada B sütunu ana tasnif kriteri ve A sütunu ise tali tasnif kriteri olarak kullanılmak suretiyle değerler küçükten büyüğe dpğru (artan) sıralanmışlardır.

	A	B	C	D
1	George	Washington	1789-1797	
2	John	Adams	1797-1801	
3	Thomas	Jefferson	1801-1809	
4	James	Madison	1809-1817	
5	James	Monroe	1817-1825	
6	John Quincy	Adams	1825-1829	
7	Andrew	Jackson	1829-1837	
8				

(a) Before sort

	A	B	C	D
1	John	Adams	1797-1801	
2	John Quincy	Adams	1825-1829	
3	Andrew	Jackson	1829-1837	
4	Thomas	Jefferson	1801-1809	
5	James	Madison	1809-1817	
6	James	Monroe	1817-1825	
7	George	Washington	1789-1797	
8				

(b) After sort

Şekil 3.14: Dosya menüsü.

```
    void MainWindow::sort()
```

[9] kabul et

[10] reddet, kabul etme

```
{
      SortDialog dialog(this);
      QTableSelection sel = spreadsheet->selection();
      dialog.setColumnRange( A  + sel.leftCol(),  A  +
                  sel.rightCol());
      if (dialog.exec()) {
            SpreadsheetCompare compare;
            compare.keys[0] =
                  dialog.primaryColumnCombo->currentItem();
            compare.keys[1] =
                  dialog.secondaryColumnCombo->currentItem() - 1;
            compare.keys[2] =
                  dialog.tertiaryColumnCombo->currentItem() - 1;
            compare.ascending[0] =
                  (dialog.primaryOrderCombo->currentItem() == 0);
            compare.ascending[1] =
                  (dialog.secondaryOrderCombo->currentItem()==0);
            compare.ascending[2] =
                  (dialog.tertiaryOrderCombo->currentItem()==0);
            spreadsheet->sort(compare);
      }
}
```

`sort()` fonksiyonundaki kod felsefesi `goToCell()` fonksiyonundaki koda benzer:

- Diyaloğu yığında oluşturuyoruz ve ilk ayarlamaları yapıyoruz.
- `exec()` fonksiyonunu çağırarak diyaloğu görüntülüyoruz.
- Kullanıcı *Tasnif* düğmesine basınca kullanıcı tarafından girilmiş değerleri alıp gerekeni yapıyoruz.

`compare`[11] nesnesi birinci, ikinci ve üçüncü tasnif anahtarlarını ve sıralarını ihtiva eder. (`SpreadsheetCompare` sınıfının tanımını bir sonraki bölümde göreceğiz.) Bu nesne `Spreadsheet::sort()` tarafından iki sırayı kıyaslamak için kullanılır. keys[12] listesi anahtarların sütun numaralarına muhtevidir. Mesela, seçilen alan C2 den E5 e kadar ise bu durumda C sütunu listenin 0 pozisyonunda yer alır. ascending[13] listesi her bir anahtar ile mütaalik olan sırayı `bool` türü (menfi=0,müsbet=1)bir değer olarak içerir. `QComboBox::currentItem()` fonksiyonu hali hazırda seçilmiş olan ferdin indeksini getirir ki bu indeks sıfırdan başlar. İkinci ve üçüncü anahtarlar içim şeçilmiş olanin indeksinden bir çıkarıyoruz taki listede mevcut olan Hiç ferdini gözardı etmemiş olalım.

`sort()` fonksiyonu işini görür ancak çok titizdir. Tasnif diyaloğunun fırıldak kutuları ile oluşturulduğunu ve "Hiç" ferdlerini içerdiğini varsayar. Bu demek oluyorki, Tasnif diyaloğunu yeniden tasarlayacak olursak bu kodu yeniden yazmamız gerekecek. Bu yaklaşım sadece bir yerden öaşrılan diyaloglar için kafi iken, proğramın bir çok yerinden çağrılan bir diyalog için proğramcının korkulu rüyası olur.

---

11 mukayese et, kıyas et, karşılaştır

12 anahtarlar

13 artan

Daha sağlam bir yaklaşım `SortDialog` sınıfını daha kapsamlı yapıp onun `SpreadsheetCompare` nesnesini kendisini oluşturmasını sağlamaktır ki bu nesne dğyaloğu şagıran tarafından kullanılır. Bu `MainWindow::sort()` fonksiyonunu fevkalade basite indirger:

```
void MainWindow::sort()
{
      SortDialog dialog(this);
      QTableSelection sel = spreadsheet->selection();
      dialog.setColumnRange( A  + sel.leftCol(),  A  +
                              sel.rightCol());
      if (dialog.exec())
            spreadsheet->performSort(dialog.comparisonObject());
}
```

Bu yaklaşım gevşek bir şekilde bir birine başlanmış azalar teşkil eder ki kanatimizce hemen hemen bütün ortamlarda birden fazla yerden çaşrılacak diyaloğlar için bu en doğru bir yaklaşımdır.

Daha radikal bir yaklaşımda `SortDiyalog` nesnesini oluştururken ona `Spreadsheet` nesnesini bir timsalini göndermek suretiyle bu diyaloğun Spreadsheet ile direk temasını sağlamaktır. Bu `SortDialog` sınıfınının umumiliğini kısmende olsa kaybetmesine sebep olur çünkü o sadece belirli bir alet ile öalışır olur ancak `SortDialog::setColumnRange()` fonksiyonuna gerek olmadığından kodu iyicr basitleştirir. `MainWindow::sort()` fonksiyonu şu hali alır:

```
void MainWindow::sort()
{
      SortDialog dialog(this);
      dialog.setSpreadsheet(spreadsheet);
      dialog.exec();
}
```

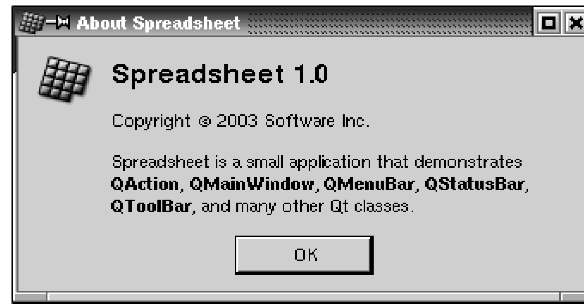
Bu yaklaşım bir bakıma ilk yaklaşımın zıddıdır: Diyaloğu çağıranın diyalog hakkında detaylı bilgiye sahıp olmasına mukabil, diyaloğun kendisini çağıranın sağladığı data hakkında detaylı bilgiye sahip olması zorunluluğu var. Bu yaklaşım diyaloğun yapılan değişiklikleri ani olarak yansıtması gerektiği durumlarda tercih edilir. Nasılki ilk yaklaşimda çağırıcının kodu titiz ise bu yaklaşımdada çağıranın data yapısı değişmesi durumunda diyalog çalışmaz hale gelir.

Bazı proğramcılar bir metudu seçip sürekli o metodu kullanırlar. Bu proğramın içerisindeki diyalogların helsinin aynı yaklaiımı takip etmelerine binaen kolaylık olur ancak dişer metodların avantajlarından faydalanılamaz. Hangi methodun yada yaklaşımın kullanılacağı her bir dıyaloğun kendine has kullanım ve görevleri göz önünde bulundurularak deçilmelidir.

Bu kısmı *Proğram Hakkında* kutusu ile tamamlayacağız. *Ara* ve *Hücreye Git* diyaloglarında olduğu gibi hususi bir diyalog oluşturabilirdik ancak bu Proğram Hakkında kutularının içerikleri muhtelif olabileceği için Qt basit bir çözüm sunar.

```
void MainWindow::about()
{
   QMessageBox::about(this,trUtf8("Elektronik Çizelge Hakk nda"),
      trUtf8("<h2>Elektronik Çizelge 1.0</h2>"
      "<p>Her Hakk  Mahfuzdur &copy; 2004 Software Inc."
      "<p>Elektronik Çizelge pro ram  ba ta  "
      "<b>QAction</b>,<b>QMainWindow</b>, "
      "<b>QMenuBar</b>, <b>QStatusBar</b>, "
      "<b>QToolBar</b>, s n flar  olmak üzere
      " bir çok Qt s n f n n  kullan m n   gösterir."));
}
```

Proğram Hakkında kutusu *sakin* `(static)` `QmessageBox::about()` fonksiyonunu çağırmak suretiyle oluşturulur. Bu `QmessageBox::warning()` fonksiyonuna çok benzer. Aralarındaki fark about() atasının simgesini kullanır ancak warning() normal "warning[14]" simgesini kullanır.



Şekil 3.15: Dosya menüsü.

Şu ana kadar kolaylık sağlayan `QMessageBox` ve `QFileDialog` sınıflarında mevcut bir kaç statik fonksiyonu kullandık. Bu fonksiyonlar diyalogları oluşturup, ilk ayarlarını yaptıktan sonra `exec()` fonksiyonu vasıtasıyla ekranda görünmelerini sağlarlar. Her ne kadar bir ünceki metod kadar kolay olmasada kendimiz `QMessageBox` veta `QFileDialog` aletleri oluşturup `exec()` veya hatta `show()` fonksiyonlarını çağırarak bunları görüntüleyebiliriz.

Proğram Ayarlarının Muhafaza Edilmesi

`MainWindow` un yapıcısında `readSettings()` fonksiyonunu çağırmak suretiyle proğram ayarlarını yüklüyoruz. Aynı şekilde `closeEvent()` fonksiyonu işerisisnde `writeSettings()` fonksiyonunu şağırmak suretiyle ayarlaeı diske kaydediyoruz. Bu iki fonksiyon yazmamız gereken ve `MainWindow` tarafından ihtiyaç duyulan son iki fonksiyondurlar.

Bizim `MainWindow` içerisisnde ayarları yazmak ve okumak için seçtiğimiz yaklaşım mümkün olan metodlardan bir tanesidir. Proğram çalışırken proğramın herhangi bir yerinden bir `QSettings` nesnesi oluşturulmak suretiyle ayarlar aranabilir yada

[14] tenbih,ihtar,uyarı

değiştirilebilir.

```
void MainWindow::writeSettings()
{
        QSettings settings;
        settings.setPath("software-inc.com", "Spreadsheet");
        settings.beginGroup("/Spreadsheet");
        settings.writeEntry("/geometry/x", x());
        settings.writeEntry("/geometry/y", y());
        settings.writeEntry("/geometry/width", width[15]());
        settings.writeEntry("/geometry/height", height[16]());
        settings.writeEntry("/recentFiles[17]", recentFiles);
        settings.writeEntry("/showGrid[18]", showGridAct->isOn());
        settings.writeEntry("/autoRecalc[19]", showGridAct->isOn());
        settings.endGroup();
}
```

`writeSettings()` fonksiyonu ana penceredin geometrisini (mevkisi ve ebatları), son zamanda açılmış olan dosyaların listesi, *Izgarayı göster* ve *Automatik olarak yeniden hesapla* seçeneklerinin durumunu kaydeder.

`QSettings` proğramın seşeneklerine işletim sistemine bağlı olarak değişik yerlerde muhafaza eder. Windows altında, system registry sini kullanır; Unix altında, text dosyalarına yazar; Mac OS X altında ise Carbon seçenekler (preferences) API kullanır. `setPath()` çağrısı `QSettings` e firmanın ismini (Internet adresi olarak) ve proğram ismini bildirir. Bu bilgi işletim sistemine göre seçenekler için müsait bir ter bulmada kullanılır.

`QSettings` ayarları anahtar ve o anahtar ile ilgili olan degeri bir çift olarak kaydeder. Anahtar dosya yoluna benzer ve daima ptoğramın ismi ile başlamalıdır. Mesela, `/Spreadsheet/geometry/x` ve `/Spreadsheet/showGrid` birer geçerli anahtardırlar. (`beginGroup()` fonksiyonuna yapılan çağrı her defasında `/Spreadsheet` yazmaktan bizi kurtarıyor.) Değer `int`, `bool`, `double`, `QString` veya `QstringList` olabilir.

```
void MainWindow::readSettings()
{
        QSettings settings;
        settings.setPath("software-inc.com", "Spreadsheet");
        settings.beginGroup("/Spreadsheet");

        int x = settings.readNumEntry("/geometry/x", 200);
        int y = settings.readNumEntry("/geometry/y", 200);
        int w = settings.readNumEntry("/geometry/width", 400);
        int h = settings.readNumEntry("/geometry/height", 400);
        move(x, y);
        resize(w, h);
```

15 genişlik

16 yükseklik

17 son zamanda açılmış olan dosyalar

18 ızagarayı göster

19 (gerekli ise) automatik olarak yeniden hesapla

```
        recentFiles = settings.readListEntry("/recentFiles");
        updateRecentFileItems();
        showGridAct->setOn(
              settings.readBoolEntry("/showGrid", true));
        autoRecalcAct->setOn(
              settings.readBoolEntry("/autoRecalc", true));

        settings.endGroup();
  }
```

`readSettings()` fonksiyonu `writeSettings()` fonksiyonu tarafından muhafaza veya kaydedilen ayarları okur. "Read" fonksiyonlarının ikinci argumanı ayarın bulunmaması durumunda kullanılacak olan varsayılan değerdir. Proğram ilk çalıştırıldığında bu varsayılan değerler kullanılırlar.

Böylece Spreadsheet proğramının `MainWindow` unun ifasını böylece tamamlamış olduk. Gelen kısımda Spreadsheet proğramını değiştirerek nasıl birden fazla dosya ile çalışabilir hale getirileceğini göreceşimiz gibi birde takdim penceresinin nasıl ifa edileceğini göstereceğiz. Ancak takdim penceresinin detayları gelen bölümde tamamlanacaktır.

Müteaddid Dosyaların

Şimdi artık Spreadsheet proğramının `main()` fonksiyonunu yazabiliriz:

```
#include <qapplication.h>
#include "mainwindow.h"
int main(int argc, char *argv[])
{
QApplication app(argc, argv);
MainWindow mainWin;
app.setMainWidget(&mainWin);
mainWin.show();
return app.exec();
}
```

Bu proğramın `main()` fonksiyonu şu ana kadar gördüklerimizden biraz farklıdır: `MainWindow` un nesnesini `new` kullanarak oluşturmak yerine yığında oluşturduk. Bu durumda `MainWindow` nesnesi maın() fonksiyonu sona erdiğinde otomatik olarak imha edilir.

Hali hazırdaki `main()` fonksiyonu sadece bir pencere oluşturu ki buda kullanıcının sadece bir dosyayı açmasına imkan sağlar. Müteaddid dosyaları aynı anda düzenlemek yada edit etmek istiyorsak Spreadsheet proğramını bir kaçdefa başlatabiliriz. Ancak bir proüram iöerisinde birden fazla dosyanın birden açılabilmesi aynı proğramın birden fazla kopyasının aynı anda koşturulmasına müreccehdir.

Spreadsheet proğramını değiştireceğiz taki müteaddid dosyalar ile çalışabilsin. Evvela biraz farklı Dosya menüsüne ihtiyacımız var:

- *Dosya | Yeni* menüsüs hali hazırdaki pencereyi kapatmak yerine yeni bir ana pencere açsın.
- *Dosya | Kapat* menüsü faal olan pencereyi kapatsın.
- *Dosya | Çık* bütün pencereleri kapatsın.



Figure 3.16. The new File menu

Bizim orjinal *Dosya* menümüzde *Kapat* diye bir komut yoktu çünkü tek bir pencere için bu *Çık* komutu ile aynı vazifeyi görürdü. İşte yeni `main()` fonksiyonu:

```
#include <qapplication.h>
#include "mainwindow.h"
int main(int argc, char *argv[])
{
      QApplication app(argc, argv);
      MainWindow *mainWin = new MainWindow;
      mainWin->show();
      QObject::connect(&app, SIGNAL(lastWindowClosed()),
                       &app, SLOT(quit()));
      return app.exec();
}
```

`QApplication` sınıfının `lastWindowClosed()` sinyalini yine `QApplication` sınıfının `quit()` dilimine bağlıyoruz kı buda son pencere kapanır kapanmaz proğramın sona ermesine neden olur.

`MainWindow` nesnesini `new` kullanarak oluğturduk. Bu gayet makuldur çünkü müteaddid pencerelerin mevcut olması durumunda işimizin bittiği pencereyi bellekte yer açmak için `delete` vasıtası ile silebiliriz. Bu mesele proğramın yanlız bir pencereye sahıp olması durumunda ortaya çıkmaz.

İşte `MainWindow::newFile()` diliminin son hali:

```
void MainWindow::newFile()
{
      MainWindow *mainWin = new MainWindow;
      mainWin->show();
}
```

Aslında burada yaptığımız sadece bir `MainWindow` nesnesi oluşturmaktır. Dikkate edilirse yeni nesneye ait harhangi bir timsali elimizde bulundurmaya ihtiyacımız yok çünkü Qt bütün pencereleri takip eder.

These are the actions for *Close* and *Exit*:

```
closeAct = new QAction(tr("&Close"), tr("Ctrl+W"), this);
connect(closeAct, SIGNAL(activated()),
            this, SLOT(close()));

exitAct = new QAction(tr("E&xit"), tr("Ctrl+Q"), this);
connect(exitAct, SIGNAL(activated()),
           qApp, SLOT(closeAllWindows()));
```

QApplication in closeAllWindows() dilimi proğramın butün pencelerini, pencerelerden hıç birisi kapatılmayı reddetmemek şartıyla, kapatır. Burada bizim istediğimiz davranışta budur. Kaydedilmemiş olan değişiklikler için kafa yormamıza gerek yoktur çünkü bu MainWindow::closeEvent() tarafından herbir pencere kapatılınca halledilmektedir.

Şu aşamada proğram her ne kadar müteaddid dosyalarla çalışabilir kebiliyette imiş gibi görünsede, malesef halledilmesi gereken önemli bir husus daha var: Kullanıcı yeni pencereler açıp kapatmaya devam ederse bir müddet sonra bilgisayarın belleği tamamen dolacaktır! Bunun sebebi ise newFile() içerisisnde MainWindow oluituruyoruz ancak pencere kapatıldığında imha edip işgal ettiği belleği temizlemiyoruz. Kullanıcı bir pencereyi kapattığında normalde pencere saklanır ve bellekte yer kaplamaya devam eder. Çok sayıda pencerenin açılıp kapatılması halinde hafızanın temizlenmemesi problem teşkil edebilir.

Çözüm yapıcıya WDestructiveClose seçeneğini eklemektir:

```
MainWindow::MainWindow(QWidget *parent, const char *name)
      : QMainWindow(parent, name, WDestructiveClose)
{
      ...
}
```

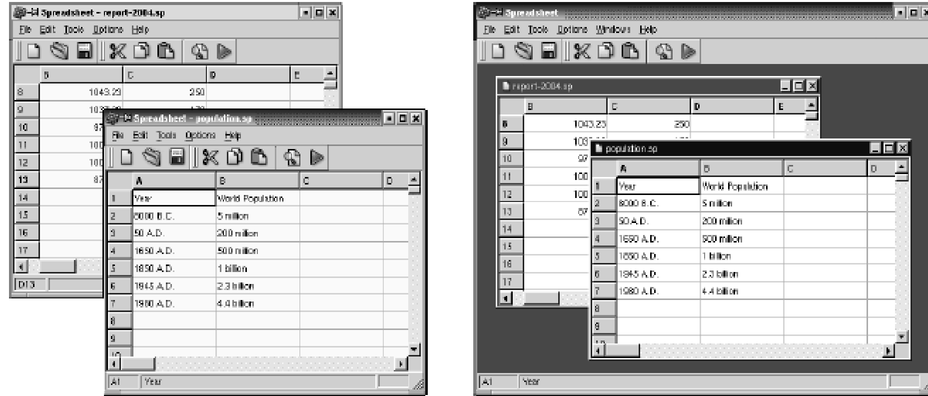
Bu seçenek Qt den pencere kapatıldığında onu silmesini ister. WDestructiveClose seçeneği QWidget yapıcısının argümanlarında sadece bir tanesidirki bu tür seçenekler pencerenin davranışını belirlerler.

Hafıza yada bellekteki sızıntılar yada kaçaklar bizim alakadar olmamız gereken meselelerden sadece bir tanesidir. Bizim tasarladığımız orjinal proğram sadece bir pencerenin olacağı varsayımından hareket ederek planlanmıştı. Birden fazla pencerenin mevcudiyeti her bir pencereye has geçmişte açılan dosyalar listesi ve seçeneklerinin olmasını gerektirir. Aslında geçmişte aöılmış dosya listesi prüğrama ait olmalıdır, her bir pencereye değil. Bunu recentFiles değişkenini sakin (statik) yapmak suretiyle başarabiliriz. Bu demek oluyorki bu deüişken bütün pencereler tarafından paylaşılacaktır. Ancak updateRecentFileItems() çağırarak geçmişte açılmış *Dosya* listesini güncelleştirmek istediğimiz vakit bunu bütün ana pencereler için yapmayılıyız. İşte bunu yapacak kod:

```
QWidgetList *list = QApplication::topLevelWidgets();
QWidgetListIt it(*list);
QWidget *widget;
while ((widget = it.current()))
{
      if (widget->inherits("MainWindow"))
            ((MainWindow *)widget)->updateRecentFileItems();
      ++it;
}
delete list;
```

Bu kod proğramın bütün ana pencereleri için updateRecentFileItems() fonksiyonunu çağırır. Benzeri kod, seçenekler dosyasını çok defa yüklememek için ve "ızgarayı göster", "otomatik olarak yeniden hesapla" gibi seöeneklerin pencereler arasında sınkronize edilmesi

için kullanlabilir. `QWidgetList` aslında `QptrList<QWidget>` için bir typedef dir. Bu ve benzeri muhtevi sınıfları 11. bölümde ele alacağız.



Şekil 3.17: SDI ve MDI

Her bir ana pencere başına bir dosyanın tekabul ettiği proğramlara SDI (single document interface) adı verilir. Bir diğer yaklaşımda MDI (multiple document interface) metodudur kı bu methodda her bir ana pencere öok sayıda dosyayı aynı anda idare edebilir. Qt çalıştığı bütüm işletim sistemleri altında hem SDI hemde MDI proğramları yazmak için kullanılabilir. Şekil 3.17 de preadsheet proğramını hem SDI hemde MDI olarak şalıştığı haliyle görebilirsiniz.

Takdim Ekranları

Çoğu proğramlar başlarken takdim ekranı görüntülerler. Bazı proğramcılar bunu proğrmın başlaması için uzun zaman gerektiğini kullanıcıdan gizlemek için yaparken bazılarıda raklam maksadı ile yaparlar. At poğramlarına `QSplashScreen` sınıfını kullanarak takdim ekranı eklemek gayet kolaydır.

`QSplashScreen` sınıfı proğram başlamdan önce genelde bir resim olan takdim ekranını görüntüler. Aynı zamanda resim üzerine proğramın yükleme durumunu güsteren yazıda yazılabilir. Genelde takdim ekranı ike alakalı kod `main()` fonksiyonunun içinde `Qapplication::exec()` fonksiyonuna yapılan çağrıdan çnce yer alır.

Aşağıdaki `main()` fonksiyonu `QSplashScreen` sınıfını kullanrak takdim ekranının proğram başlarken görüntüler ve network bağlantıları ve azaların (modules) yüklenme durumu hakkında kullanıcıyı bilgilrndirir.

```
int main(int argc, char *argv[])
{
   QApplication app(argc, argv);
```

```
    QSplashScreen *splash =
       new QsplashScreen(QPixmap::fromMimeSource("splash.png"));
    splash->show();
    splash->message(
       Qobject::trUtf8("Ana pencere düzenleniyor..."),
       Qt::AlignRight | Qt::AlignTop, Qt::white);
    MainWindow mainWin;
    app.setMainWidget(&mainWin);
    splash->message(QObject::tr("Azalar yükleniyor..."),
       Qt::AlignRight | Qt::AlignTop, Qt::white);
    loadModules();
    splash->message(QObject::tr("Ba lant lar  kuruluyor..."),
                              Qt::AlignRight | Qt::AlignTop,
Qt::white);
    establishConnections();
    mainWin.show();
    splash->finish(&mainWin);
    delete splash;
    return app.exec();
}
```



Şekil 3.18: SDI ve MDI

Böylece Spreadsheet proğramının kullanıcı arabirimi kısmını tamamlamış oluyoruz. Bir sonraki bölümde elektronik çizelge için gerekli kaynak kodu ve diğer gerekli kısımları tamamlayacağız.